Activity Indicator y addFavoritos

En este ejercicio se actualiza el estado del *Store* para almacenar las excursiones "favoritas" del usuario. Además, se añade un símbolo loader para indicar que se está cargando la página.

1. Activity Indicator

Inicialmente se añade el loader para indicar que se está cargando la app. Esto se realiza con un *Activity Indicator* de React Native.

Se crea ahora el componente *IndicadorActividad*, que será definido en el fichero *IndicadorActividadComponent.js*, dentro de la carpeta *componentes*. El código de dicho fichero será el indicado en el guion.

El código es sencillo compuesto por el *Activity Indicator,* el texto que indica "En proceso…" y un stylesheet para darle estilo.

Se puede consultar el estado de la variable ***isLoading***, seremos capaces de determinar si la carga está en proceso o ya ha terminado.

Por ejemplo, en el componente *Home* se puede consultar el estado de la variable isLoading para saber si está cargando tal y como se observa a continuación:

```
        <RenderItem item={this.props.excursiones.excursiones.filter((excur
sion) => excursion.destacado)[0]}
                    isLoading={this.props.excursiones.isLoading}
                    errMess={this.props.excursiones.errMess}

                />
```

En el componente funcional RenderItem se comprueba si la variable isLoading es true o no, si es true mostrará el laoder.

Esto también se implementa en los componentes QuienesSomos y Calendario, aunque tampoco ahora tiene mucho sentido ya que nosotros no lo vamos a poder ver.

2. Guardar favoritos en el Store

Ahora se va a modificar la funcionalidad de favoritos agregándola a Redux para que mantenga el estado, aunque se salga del componente DetallesExcursion.

Lo primero se van a definir dos nuevos action types en el archivo ActionTypes.js:

- (POST_FAVORITO) se empleará para subir la información al servidor.
- (ADD_FAVORITO) será ejecutada una vez completada la primera.

Se crea ahora un nuevo reducer favoritos.js en la carpeta de redux con el código indicado en el guion. Observando el código solo se implementa respuesta para la acción ADD_FAVORITO.
Este reducer espera un payload que será ExcursionId.

En el ActionCreators.js se añade el código necesario tal y como está indicado en el guion.

La función postFavorito() introduce un retardo de dos segundos, para "simular" para lanzar la función addFavorito que devuelve la acción de tipo ADD_FAVORITO.

Ahora se debe modificar el código de DetalleExcursionComponent.js:

```
import { postFavorito } from '../redux/ActionCreators';

const mapStateToProps = state => {
    return {
      excursiones: state.excursiones,
      comentarios: state.comentarios,
      favoritos: state.favoritos
    }
  }

  const mapDispatchToProps = dispatch => ({
    postFavorito: (excursionId) => dispatch(postFavorito(excursionId)),
})
```

- Primero se importa postFavorito
- Se añade a mapStateToProps favoritos
- Se crea el método *mapDispatchToProps* para poder lanzar la función *Thunk.*
- Ahora ya se puede borrar el constructor ya que ahora no es necesario porque no se almacena en el estado sino con Redux.
- Finalmente se actualiza RenderItem como se había hecho en commits anteriores.

```
        <RenderExcursion

            excursion={this.props.excursiones.excursiones[+excursionId]}

            favorita={this.state.favoritos.some(el => el === excursionId)}

            favorita={this.props.favoritos.some(el => el === excursionId)}

            onPress={() => this.marcarFavorito(excursionId)}

        />
```

- No se debe olvidar actualizar la exportación a través de connect().

```
export default connect(mapStateToProps, mapDispatchToProps)(DetalleExcurs
ion);
```

Para terminar, se actualiza el ConfigureStore.js